Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2343 — Arquitectura de Computadores 2019-1

Tutorial 2 – Vivado – VHDL

Tutorial de VHDL para vivado

1 Introducción:

VHDL es un lenguaje que describe circuitos digitales, no es un lenguaje de programación, donde lo que se ejecuta es una serie de pasos secuenciales. Aquí se emula conexiones de hardware.

Este tutorial tiene como objetivo familiarizarse con este nuevo lenguaje, aprendiendo conceptos básicos del mismo, y como se relaciona con Vivado.

Por lo anterior **es necesario** haber completado los dos tutoriales pasados.

2 Elementos básicos de VHDL

Si bien como lenguaje de modelación de comportamiento de circuitos eléctricos posee una serie de elementos, con los que trabajaremos para el desarrollo del curso este semestre son:

1. Librerías
2. Entidades
3. Componentes e Instancias
4. Señales
5. Operadores

2.1 Librerías:

En VHDL existen diversas librerías con distintas funcionalidades. Para este curso la librería que ocuparemos será:

```
library IEEE; -- includes
use IEEE.STD LOGIC 1164.ALL;
```

Para trabajar todo archivo que ocupen debe tener estas primeras líneas de código para importar el uso de esta librería.

2.2 Entidades:

Corresponden a la la interfaz de un bloque de hardware, esta describe los puertos que tiene un circuito y su comportamiento interno. Puede tener instancias de otro componentes dentro.

Ejemplo

En este ejemplo describiremos un circuito correspondiente a un Half-Adder. Al describir una identidad debemos definir sus inputs con **in** y los outputs con **out**.

```
library IEEE; -- includes
use IEEE.STD LOGIC 1164.ALL;


entity HA is -- declaracion de la entidad
    Port ( a : in std logic ; -- definicion de puertos
           b : in std logic ;
           s : out std logic ;
           c : out std logic );
end HA;

architecture Behavioral of HA is
-- declaraciones y definiciones
begin
   -- conexiones e instancias
end Behavioral;
```

NOTA: Este ejemplo no esta completo, y no representa un Half-Adder. Se completará más adelante

2.3 Componentes e Instancias:

Para utilizar otras piezas de hardware dentro de una entidad necesitamos **instancias**. Para instanciar otras entidades, utilizamos los **componentes**. Estos dan la información sobre que entra y sale, que tiene que ser la misma que fue definido en su entidad

Ejemplo

Por lo visto en clases, sabemos que Full Adder esta compuesto de dos Half Adder. Por tanto necesitaremos dos **instancias**, Por lo que dentro del código de la entidad que representará el Full Adder añadiremos el **componente** Half Adder para después poder instanciarlo.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FA is
    Port ( a  : in std_logic;
           b  : in std_logic;
           ci : in std_logic;
           s  : out std_logic;
           c  : out std_logic);
end FA;

architecture Behavioral of FA is
-- declaraciones y definiciones
```

```
-- Aquí definimos el componente del Half Adder
component HA
    Port ( a  : in std_logic;
           b  : in std_logic;
           s  : out std_logic;
           c  : out std_logic);
    end component;
-- Notar que tiene los mismos in y out definidos en su entidad

-- Estas son las señales serán explicadas más adelante
signal s1 : std_logic;
signal c1 : std_logic;
signal c2 : std_logic;

begin
-- conexiones e instancias

--aquí se ocupa un operador or tambien esto será explicado más adelante
c <= c1 or c2;


-- primera instancia de un Half Adder
inst_HA: HA port map(
        a      =>a,
        b      =>b,
        s      =>s1,
        c      =>c1
    );


-- primera instancia de un Half Adder
inst_HA2: HA port map(
        a      =>s1,
        b      =>ci,
        s      =>s,
        c      =>c2
    );

end Behavioral;
```

2.4 Señales:

Las señales se usan para representar cables que permite la conectividad entre partes del circuito (en el ejemplo anterior permite la conectividad entre los Half Adder y la entidad Full Adder).

Aunque también sirve como almacenamiento, de la forma:

```
signal s3 : std_logic_vector (15 down to 0);
```

Esto representa una señal de un vector de 16 bits, lo que también puede ser expresado como un bus de datos o 16 cables de un bit expresado en una señal.

2.5 Operadores:

VHDL cuenta con los operadores lógicos and, or, xor, not y de concatenación .

Ejemplo

Tomando el primer ejemplo correspondiente a un Half-Adder, agregaremos las compuertas lógicas faltantes para completar el circuito. Como se vio en clases el output "s" será "xor" entre el input "a" y el input "b", mientras que el carry "c" será el resultado del "and" entre los dos inputs mencionados anteriormente.

Para asignar el resultado de los operadores a una señal o un output se ocupa "¡=" como se aprecia en el siguiente extracto de código:

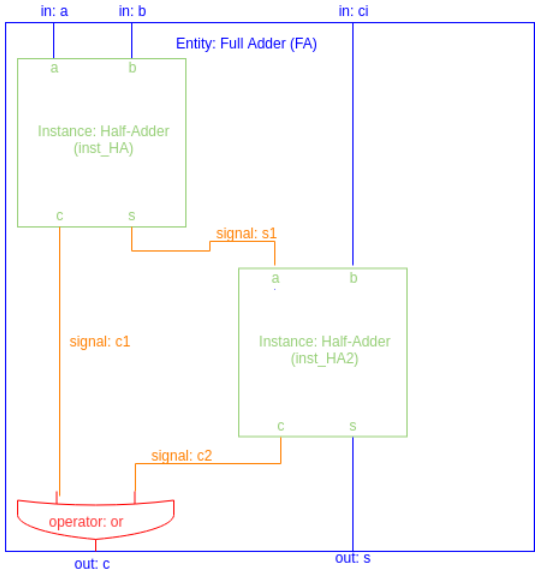
```
library IEEE; -- includes
use IEEE.STD LOGIC 1164.ALL;


entity HA is -- declaracion de la entidad
    Port ( a : in std logic ; -- definicion de puertos
           b : in std logic ;
           s : out std logic ;
           c : out std logic );
end HA;

architecture Behavioral of HA is
-- declaraciones y definiciones
begin
   -- conexiones e instancias
   s <= a xor b; -- aquí asignamos el resultado del operador "xor" entre los inputs al output s
   c <= a and b; -- aquí asignamos el resultado del operador "and" entre los inputs al output c
end Behavioral;
```

Resumiendo:

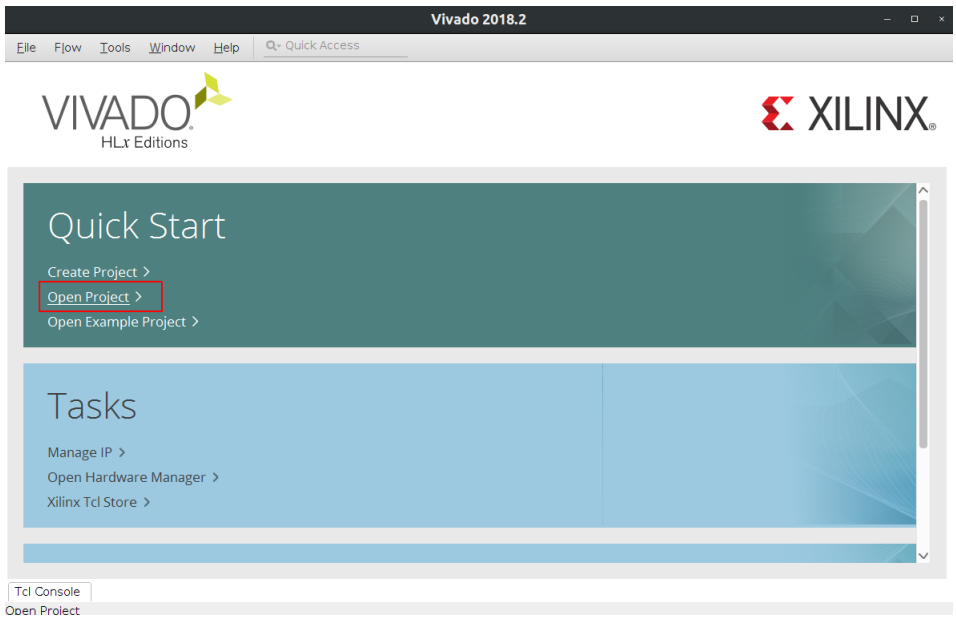
Ocupando el Full-Adder de uno de los ejemplos un resumen visual sería:



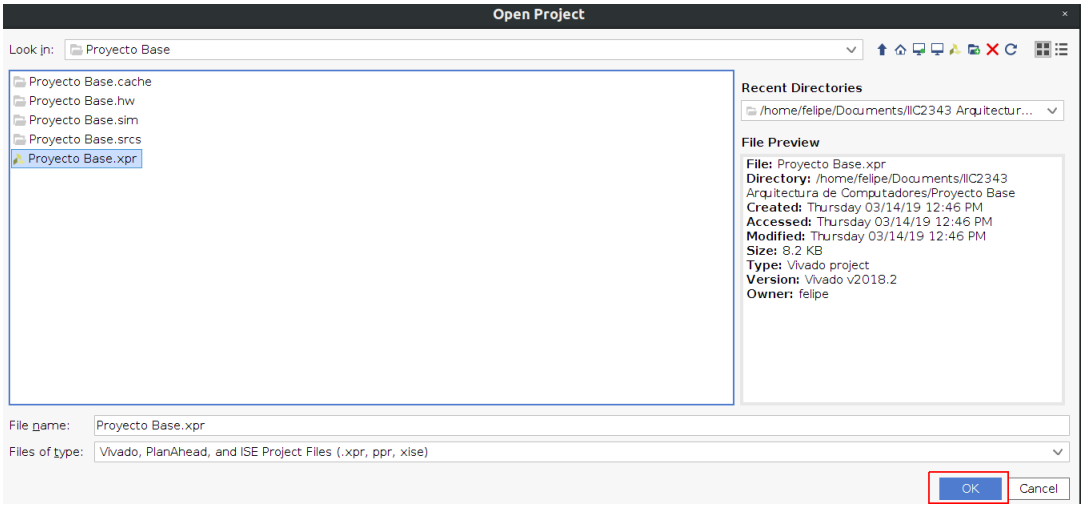
3 Usando VHDL en Vivado:

3.1 Abriendo Proyecto Base:

Como ya vimos en tutoriales pasados, abrimos Vivado y seleccionamos "Open Project":

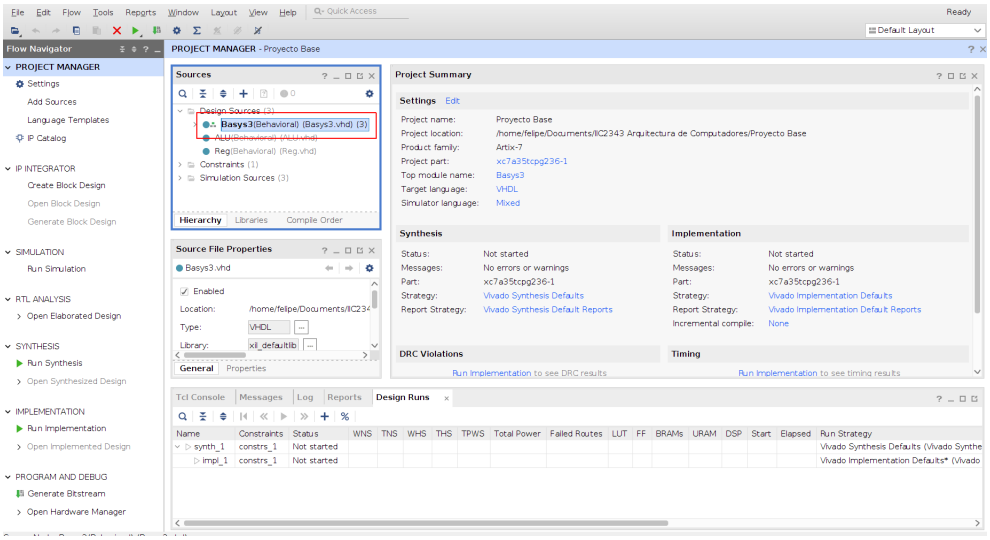


Luego buscamos el directorio donde se encuentra la carpeta "Proyecto Base" y dentro de ella seleccionamos el archivo "Proyecto Base.xpr", como se ve en la imagen, y presionamos "OK":

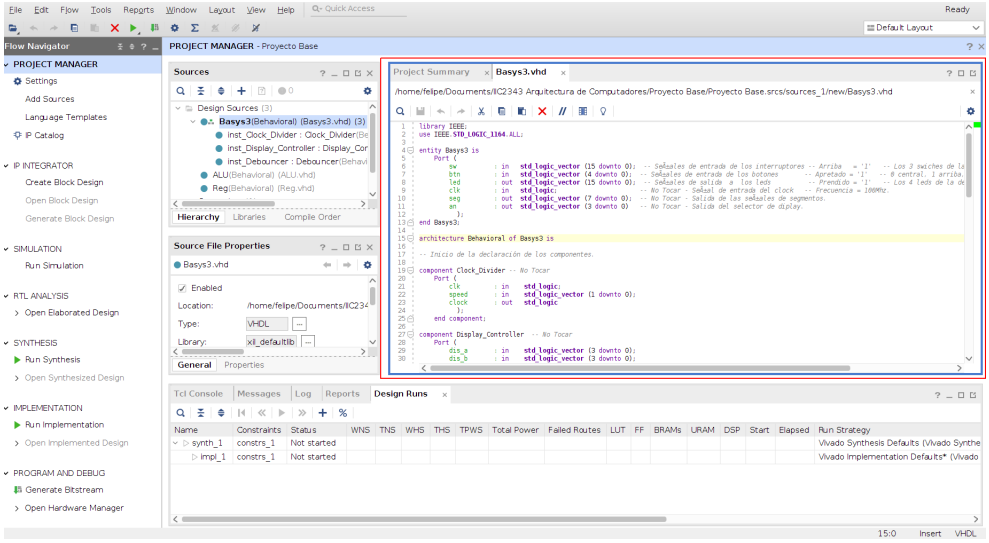


3.2 Selección de archivos:

Dentro del programa iremos al recuadro ”Sources” y presionaremos en el archivo ”Basys3”:

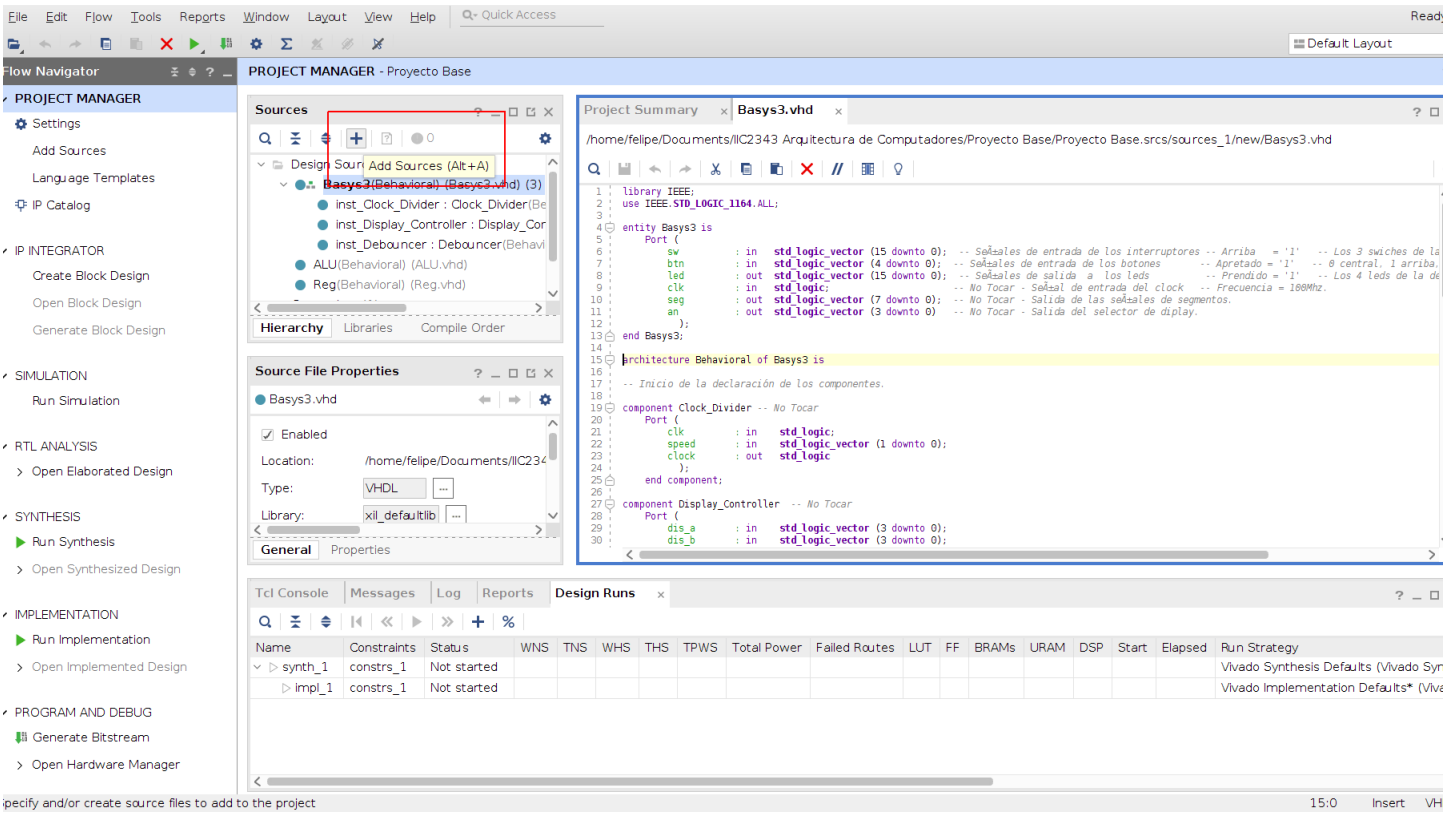


Este archivo posee todos los componentes para interactuar con la placa, se recomienda leer atentamente los comentarios dentro del mismo:

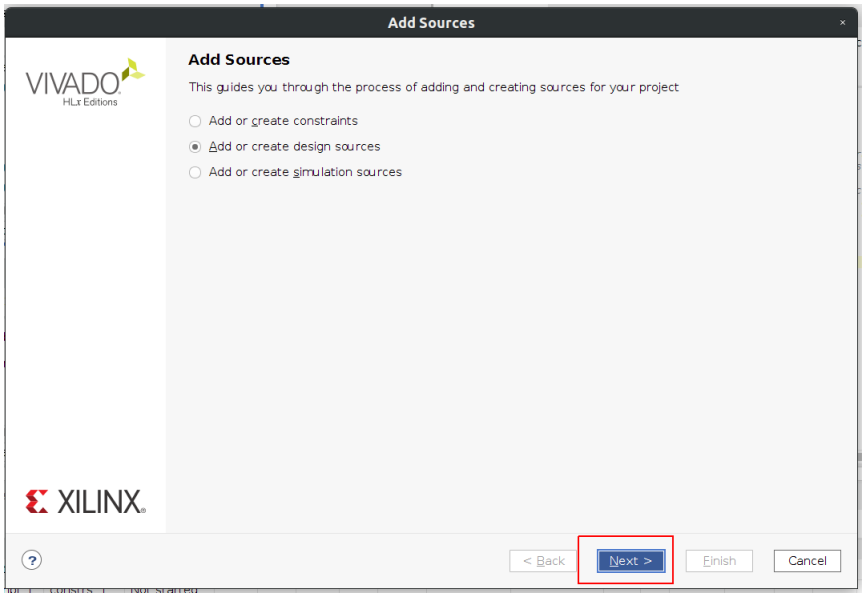


3.3 Añadiendo archivos:

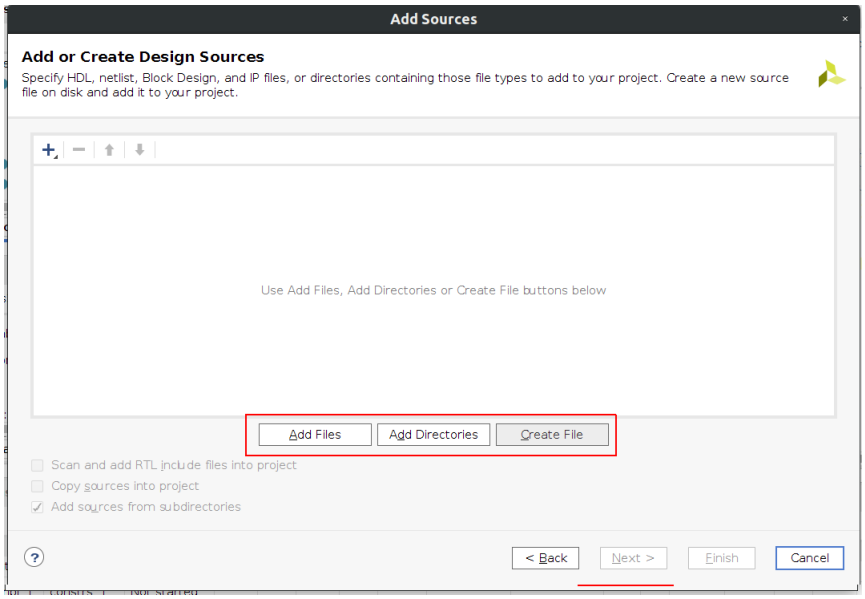
Dentro del mismo recuadro "Sources" presionaremos el botón con un símbolo "+":



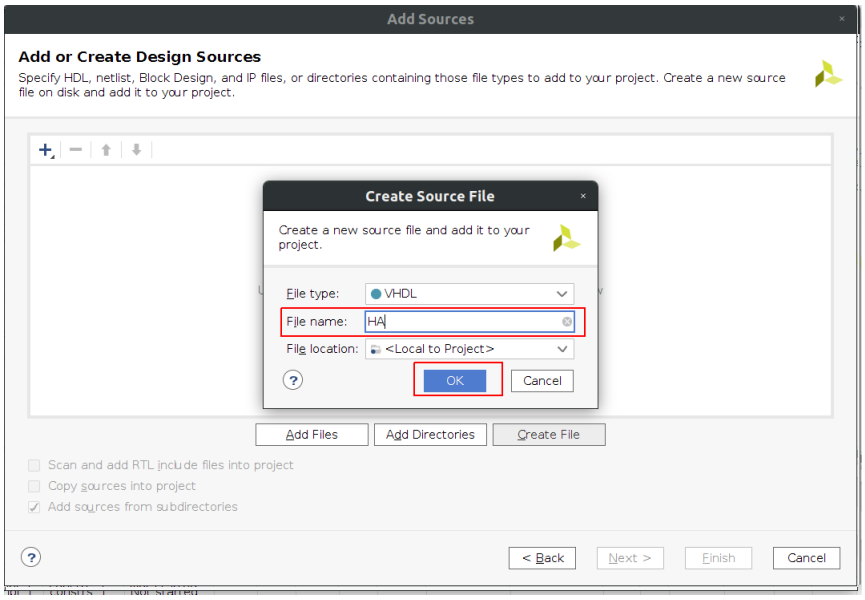
Para agregar archivos seleccionamos en "Add or create design sources" y le damos a "OK" como se ve en la imagen:



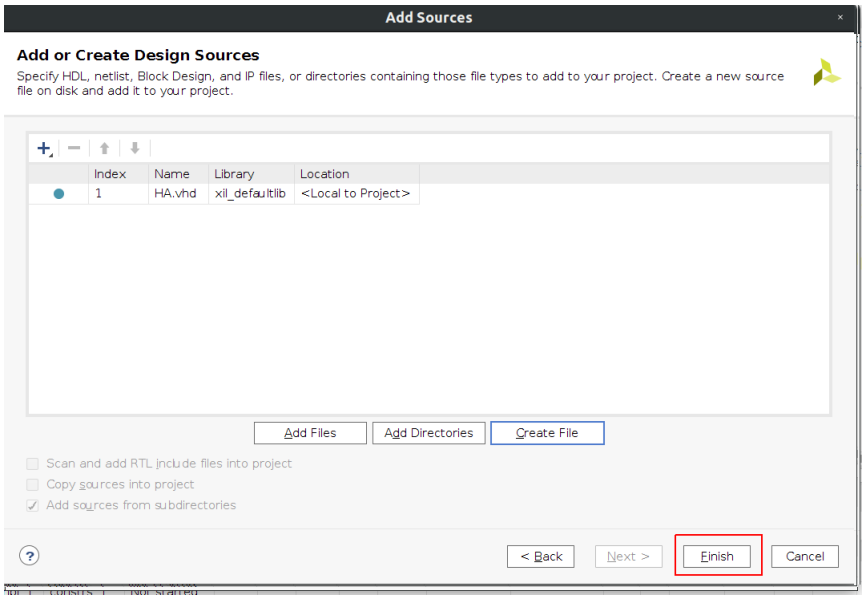
Luego la ventana nos redirigirá a si queremos añadir archivos o crear archivos. Por ahora en este tutorial nos enfocaremos en "Create File" para crear archivos, sin embargo es importante recalcar que se puede escribir código en VHDL **sin tener Vivado instalado**, y luego pasar archivos con distintas entidades a un computador **que si tenga Vivado instalado** con "Add Files". Esto es especialmente útil para el trabajo en parejas, así no olvidar tenerlo en consideración



Siguiendo la idea de crear un archivo, nos pedirá el nombre, para el caso crearemos el archivo del Half Adder llamado "HA" y presionaremos "OK" como se ve en la imagen:

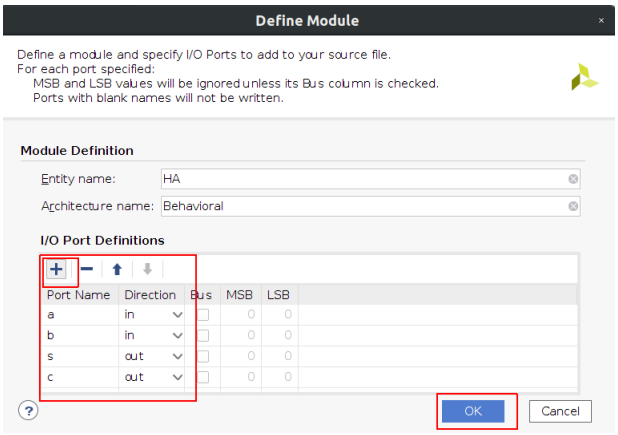


Una vez añadido y/o creado todos los archivos se presiona el botón "Finish":



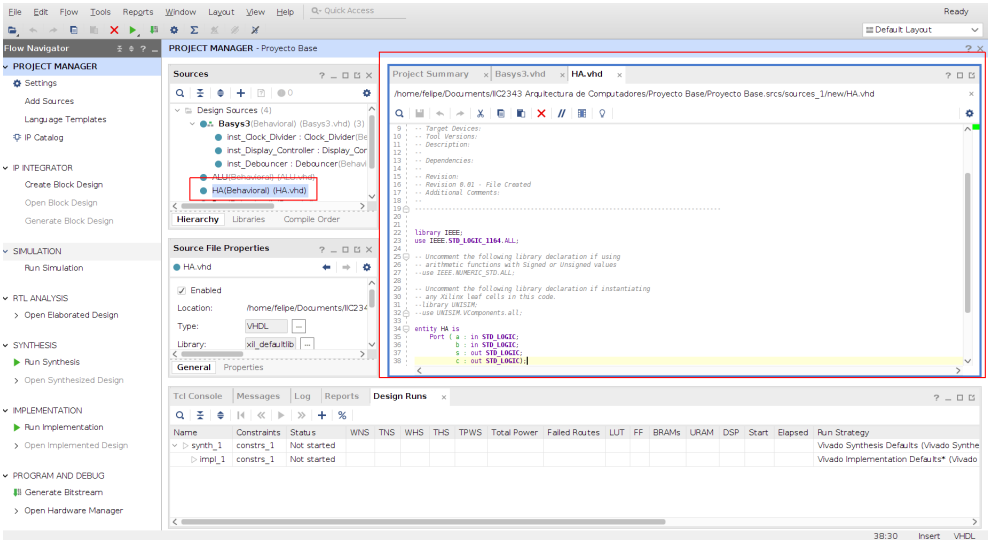
Luego se pedirá que se defina los inputs y outputs junto con la arquitectura con la que el circuito será descrito. Siempre para el desarrollo de este curso se ocupara la arquitectura "Behauvioral". Mientras que para añadir inputs y outputs solo se presiona el botón "+", luego se procede a darle un nombre en la columna "Port Name" y definir si es de salida o entrada en la columna "Direction".

Una vez definido se pulsa "OK":



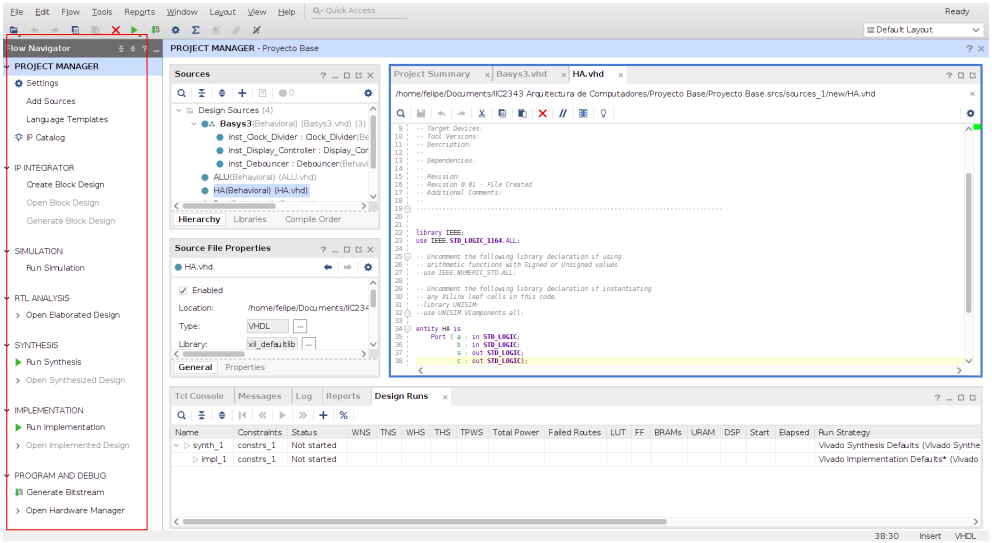
NOTA: Este procedimiento es opcional, se puede pulsar "OK" de forma inmediata y definir los inputs y outputs de la entidad manualmente después.

Una vez terminado se verá la nueva entidad con la que se podrá describir detenidamente el circuito que la compone:

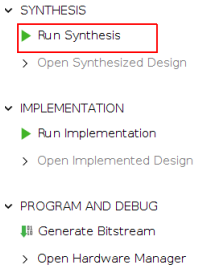


3.4 Compliando código VHDL:

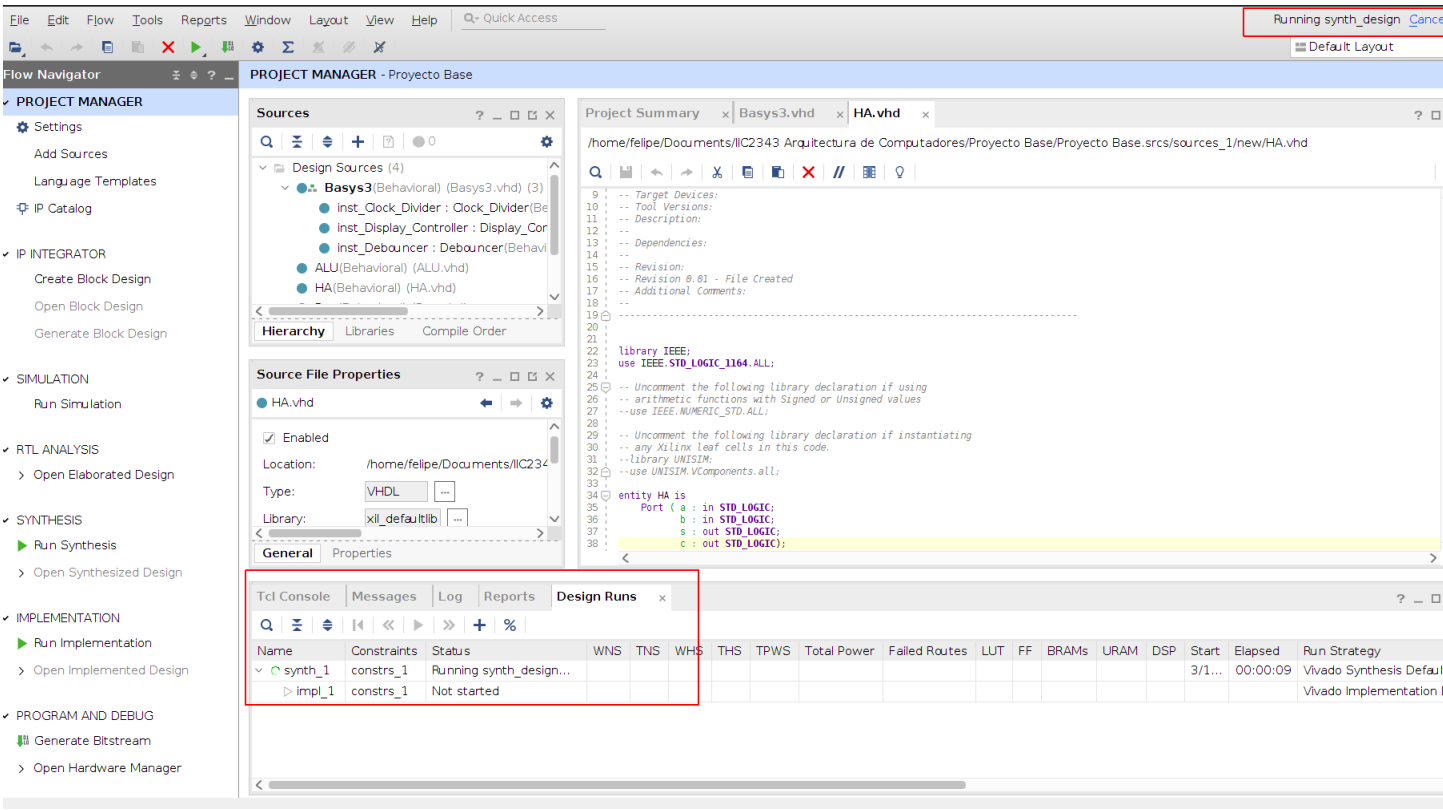
Además de ser un lenguaje descriptivo, VHDL es un lenguaje compilado, es decir, debe ser traducido por Vivado para ser transformado en un lenguaje que pueda leer una máquina, en el caso del curso la placa Basys3. Para ello nos centraremos en la columna izquierda como muestra la imagen:



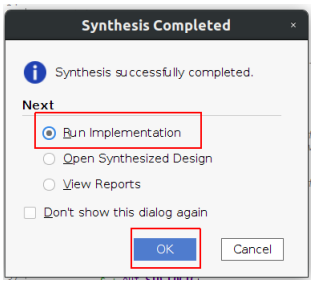
Específicamente presionaremos donde aparece "Run Syntesis":



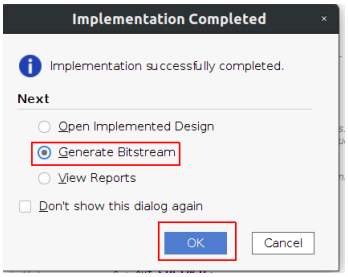
Ahora toca el proceso de esperar a que termine la síntesis del programa, se mostrará el procedimiento en el recuadro inferior del programa en la pestaña "Log" y "Messages". En caso de algún error se podrá ver en la pestaña "Reports". También se puede saber el estado de la síntesis en la esquina superior izquierda:



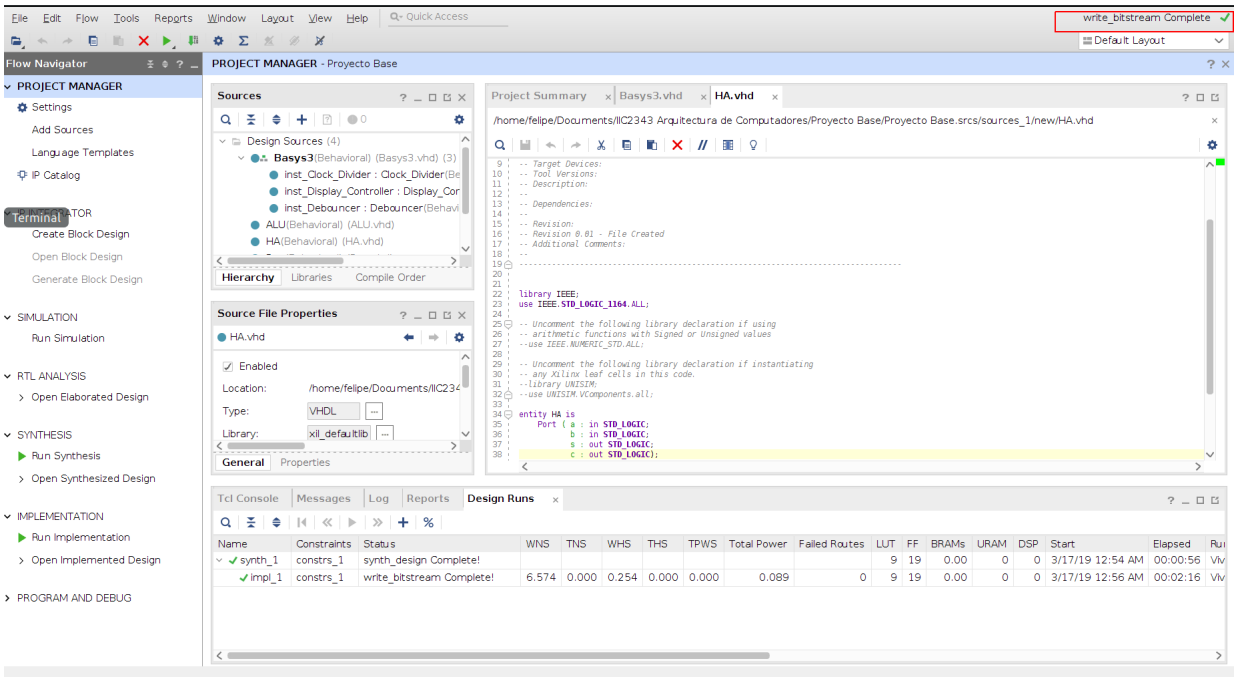
Una vez terminada a síntesis aparecerá una nueva ventana. Seleccionamos la primera opción y luego se pulsa "OK":



Ahora tenemos que esperar por otro proceso llamado implementación. Se sigue básicamente igual que la síntesis, solo que cuando termine seleccionaremos la segunda opción "Generate Bitstream" y luego "OK":



Si es que no hay ningún error, luego de esperar varios minutos, en la esquina superior izquierda debería aparecer un símbolo ✓que indica que se genero el archivo necesario para ser utilizado en la placa:



3.5 Programar la placa

Seleccionamos en "PROGRAM AND DEBUG" en la columna izquierda del programa, y luego expandimos las opciones de "Open Hardware Manager", luego presionamos en "Open Target", después "Auto-Connect", y cuando la placa este conectada "Program Device", en donde seleccionamos el archivo ".bit" con el que la placa trabajará